```java
public class Q3 {
    public static void main(String[] args) throws OverflowException, UnderflowException {
        BoundedStack<Integer> stack = new BoundedStack<>(Integer.class, 12);

        // Push elements into the stack
        int[] elements = { 2, 9, 3, 1, 8, 9, 0, 7, 8, 4, 5, 3 };
        for (int element : elements) {
            stack.push(element);
        }

        // Reverse the stack
        BoundedStack<Integer> reversedStack = reverseStack(stack);

        // Generate unique queue
        Queue<Integer> uniqueQueue = generateUniqueQueue(reversedStack);

        // Other method calls...

        // Time complexity analysis:
        // - main method: O(n)
    }

    public static <T> BoundedStack<T> reverseStack(BoundedStack<T> stack) throws OverflowException, UnderflowException {
        Queue<T> queue = new Queue<>();

        // Step 1: Pop elements from the stack and enqueue into the queue
        while (!stack.isEmpty()) {
            T element = stack.pop(); // O(1)
            queue.enqueue(element); // O(1)
        }

        BoundedStack<T> reversedStack = new BoundedStack<>(stack.getType(), stack.getStack().length);

        // Step 2: Dequeue elements from the queue and push into the reversed stack
        while (!queue.isEmpty()) {
            T element = queue.dequeue(); // O(1)
            reversedStack.push(element); // O(1)
        }

        return reversedStack;
        // Overall time complexity of reverseStack method: O(n)
    }

    public static <T> Queue<T> generateUniqueQueue(BoundedStack<T> stack)
            throws OverflowException, UnderflowException {
        Queue<T> Queuerepeated = new Queue<>();
        Queue<T> uniqueQueue = new Queue<>();

        while (!stack.isEmpty()) {
            T value = stack.pop(); // O(1)
            boolean flag = true;

            while (!uniqueQueue.isEmpty()) {
                if (value == uniqueQueue.peek()) {
                    flag = false;
                }

                Queuerepeated.enqueue(uniqueQueue.dequeue());//O(1)
            }

            if (flag == true) {
                Queuerepeated.enqueue(value);              // O(1)
            }

            while (!Queuerepeated.isEmpty()) {
                uniqueQueue.enqueue(Queuerepeated.dequeue());//O(1)
            }
        }

        return uniqueQueue;
// Overall time complexity of generateUniqueQueue method: O(n^2)
    }
}
```

Listing 1: Q3 time complexity analysis

Therefore, the overall time complexity is O(n^2)

```
public void addToCart(Tuple tuple) throws OverflowException {
    shoppingCart.push(tuple);
}

public void updateCart() throws OverflowException,
    UnderflowException {
    for (int i = 1; i < dict.size(); i++) {         // Theta(n)
        Tuple current = dict.get(i);                // Theta(1)
        Tuple prev = shoppingCart.peek();           // Theta(1)

        // Calculate prices after discount
        double prevPrice = (prev.getPrice() * (1 - prev.getDiscount
    () / 100.0));
                                                    // Theta(1)
        double currPrice = (current.getPrice() * (1 - current.
    getDiscount() / 100.0));                       // Theta(1)

        // Compare prices and update cart accordingly
        if (currPrice < prevPrice) {                // Theta(1)
            shoppingCart.pop();                     // Theta(1)
            shoppingCart.push(current);             // Theta(1)
            System.out.println((i + 1) + "th step: " + shoppingCart
     +  " as " + current.getPrice() + " X " + current.getDiscount()
     +" percent = " + currPrice + " is less then " + prevPrice);
        }
        else if (currPrice > prevPrice) {           // Theta(1)
            System.out.println((i + 1) + "th step: " + shoppingCart
     + " as " + current.getPrice() + " X " + current.getDiscount()
    +" percent = " + currPrice + " is greater then " + prevPrice);
```

```

        }

        else if (currPrice == prevPrice && current.getDiscount() > prev.getDiscount()) {                        // Theta (1)
            shoppingCart.pop();                        // Theta (1)
            shoppingCart.push(current);                // Theta (1)
            System.out.println((i + 1) + "th step: " + shoppingCart + " as " + current.getPrice() + " X " + current.getDiscount() + " percent = " + currPrice + " and " + current.getBrand() + " has a discount");

        }
        else if (currPrice == prevPrice && current.getDiscount() <= prev.getDiscount()) {                        //Theta(1)
            System.out.println((i + 1) + "th step: " + shoppingCart + " as " + current.getPrice() + " X " + current.getDiscount() +" percent = " + currPrice + " and " + prev.getBrand() + " has a bigger discount");

        }

    }
}

@Override
public String toString() {
    return shoppingCart.toString();
}

public static void main(String[] args) {
    Q4 shopping = new Q4();                            // Theta (1)

    try {

        shopping.addToCart(shopping.dict.get(0));  // Theta (1)
        System.out.println("Shoes info: " + shopping.dict);// Theta (1)
        System.out.println("1st step: " + shopping.shoppingCart);// Theta (1)

        shopping.updateCart();                     // Theta (n)
    } catch (OverflowException | UnderflowException e) {
        e.printStackTrace();
    }
}
```

Listing 2: Q4 time complexity analysis

Therefore, the overall time complexity is Theta(n)